KB/RMS: An Intelligent Assistant for Requirement Definition

Robert V. Binder
Robert Binder Systems Consulting, Inc.
Chicago, Illinois

Jeffrey J.P. Tsai
Department of Electrical Engineering and Computer Science
University of Illinois, Chicago, IL 60680

ABSTRACT

A conceptual framework and a system model for an intelligent assistant for requirement definition, KB/RMS, is presented. The requirement definition process is characterized by the Requirements Context Model. Informal and formal methods for requirement definition are considered in light of this model, which serves as the logical schema for the KB/RMS database. Conventional and knowledge-based system support for requirement definition is summarized. The use of natural language processing, a semantic model of the problem and solution spaces, domain and technology models, and inference driven augmentation, validation, and verification of the semantic model is discussed. Production of design representations from the augmented semantic model is covered.

Index Terms: Requirement definition, knowledge-based tools, knowledge acquisition, natural language processing.

I. INTRODUCTION

Software development usually begins with recognition of the user's requirements, followed by implementation of a software system which satisfies the requirements. These requirements are defined through a dialogue between users and system analysts. This process is influenced by many unstated expectations and assumptions held by both users and developers. These include perceptions about the feasibility of a software solution, the behavior of the future system, and the pre- and post-implementation dynamics of the environment.

The requirement definition process is inherently complex and ambiguous. A recent survey of 122 developers of real-time embedded systems noted that the meaning of "requirement definition" varies considerably in software engineering literature, as well as among practitioners [Wood 89]. By requirement definition, we mean "... *the process of examining user requirements and their subsequent capture into more formal representation*" [Wood 89]. This definition suggests that requirement definition has three essential aspects: 1) facilitation of group communication, 2) exploration of the problem space, and 3) elaboration of the solution model. Communication, problem space representation, and solution space representation interact and involve many tangible and abstract objects.

Although representation of tangible problem-space objects is necessary, it is not sufficient to describe the requirement definition process. If we wish to provide robust automated support for requirement definition, we need to be able to process concepts, conceptual relations, and solution objects in an integrated manner. The purpose of this paper is to establish an integrated model to explicitly represent these objects and relationships. This is the Requirements Context Model (RCM). The RCM is represented in KB/RMS.

Many existing systems support development of formal or semi-formal models of the solution space. Several systems provide problem space representation. There are no general-purpose, knowledge-based tools that represent objects in both spaces and assist software developers in translating informal requirements into solution representations (see sections III and IV).

KB/RMS is a knowledge-based requirement definition assistant. It uses rule-based inference and natural language processing to create a high level system model from a large collection of natural language statements. It produces a rough draft of a software requirement specification (SRS) from natural language statements. It also assists the software developer in refining the SRS.

KB/RMS is independent of any particular application domain and software development method. It is intended to support development of large, high-impact systems, where the application of proven software engineering methods and sound management practice is a necessity.

We discuss the Requirements Context Model in the following section. Next, current approaches and automated support are summarized. Knowledge representation in KB/RMS is presented. Finally, we outline future plans for the system.

II. THE REQUIREMENTS CONTEXT MODEL

The Requirements Context Model characterizes the requirement definition process in general, independent of a particular application domain or design method, or implementation platform. It identifies the objects in the problem and solution space necessary for development of large software systems. This model also serves as the logical schema for the KB/RMS database. Figure 1 shows the requirements context model.

2.1 The Problem Space. The *problem space* is largely the view of a person or organization that can identify and articulate the impetus for a software development project. It is comprised of entities, constraints, goals, sub-goals, and participant's perceptions and expectations. The problem space is delimited by recognition of a *conflict* that requires a resolution, a *need*, or an *opportunity* to improve a situation. The person (persons) providing this problem definition is called the *sponsor*. The sponsor formulates a *goal* that describes a desired state which is expected to resolve a conflict, meet a need, or facilitate attainment of an opportunity. The goal may involve any number of conflicts, needs, or opportunities.

Typically, the goal is general and simple. There may also be explicit *sub-goals*. A goal often has a large number of implicit *expectations* and assumptions. It has been argued that expectations and assumptions are more important than explicit goals, since there are many ways to misunderstand the sponsor's complete vision.

Goals are also influenced by *operational constraints*, which include cost budgets, developmental and operational risk preference, technological proficiency, and organizational conflicts and competencies. Externally determined technology is also an operational constraint.

The intangible aspects of the problem space must be related to things that can be represented, controlled, or modeled by a software system. These are typically tangible entities like business transactions, mechanical actuators, or people in an organization. Problem space entities can be abstract if the problem is to produce purely analytical results (e.g., a theorem proving system.) Most problems include both abstract and tangible entities. Three general types of problem space entities are *objects*, *events*, and *processes*. An object is a tangible or abstract entity that has relevance within the problem space. A process is a sequence of transformations that involves objects. An event is a particular constellation of objects or a discrete state of a process that has relevance within the problem space. Entities change states by virtue of processes.

Events are associated with change in the state of an entity.

2.2 The Solution Space. The Solution Space is the collection of knowledge, representations, and software objects that determine the software system developed to meet the goal. Goals are typically expressed as a number of specific user *requirement statements*. For example, in the statement "Reduce costs by increasing through-put by 50 percent", cost reduction is the goal, and through-put increase is the requirement statement. Requirement statements are initially conceptualized and expressed in natural language, even if later translated to a formal representation.

The task of system design is to develop implementable representations that realize the requirement statements. This task is guided by a *technical paradigm*. A technical paradigm is a collection of ideas and techniques about how to develop software systems.

A technical paradigm provides *heuristics*. For example, "Identify and describe the control thread for each input message." [Alford 85]. Each paradigm may be based on an explicit *model* of either the problem space or software systems. Examples of models are hierarchic functional decomposition, object-oriented, finite-state machine, petri-net, queuing networks, etc. Paradigms often share models. There is often a graphic or textual *notation* that supports expression and analysis of constructs in the paradigm.

The software design process should result in a system that meets the sponsor's goals. This system must interact with problem space entities. Interaction requires interfaces with and representations of these entities, as well as related algorithmic transformation and control. The technical paradigm organizes the designer's perceptions of the problem space, and provides tools to articulate and verify this perception. The *problem view* is the designer's point of view on the problem entities, as informed by the technical paradigm. The designer performs cognitive processing to place new facts into this view as information about the problem space is gathered. While the designer's perception is not tangible, it is essential to proper solution of the problem. Even if no explicit representation is produced, a problem view must be created. It would not be possible to produce any kind of useful specification without one.

Design components are concrete expressions of the problem view. They are derived by applying the heuristics, models, and notation of the technical paradigm. They include all design and specification work-products and are eventually realized in an *implementation*. An implementation is a software product that is intended to meet the sponsor's goals. Both representation and implementation are influenced by *development constraints*, which include cost limits, risk preference, technological proficiency, and organizational considerations specific to the development project.

It is not uncommon to reuse generic solutions, tailored to the system at hand. The generic, reusable parts of a technology paradigm are called *solution clichés*.

III. Approaches to Requirement Definition

A clear understanding of the essential properties of the requirement definition process is necessary for developing support tools. We take the goal to be production of a "good" Software Requirements Specification (SRS), as defined in [IEEE 84]. Existing approaches to requirement definition fall into two broad categories: informal and formal [Davis 88], [Gause 89], [Wood 89], [Wing 88], [Yaverbaum 89], [Yeh 84], [Zave 82], [Zultner 89].

Table 1 lists our assessment of the support these approaches can provide. If the approach makes meeting an SRS objective difficult, it is termed an obstacle. If the approach does not influence attainment, it is termed neutral. If the approach facilitates an SRS objective, it is termed enabling. If circumstances can significantly influence the outcome, then the most likely outcome is given first. Neither approach can facilitate all SRS goals. Even if both approaches are combined, the support for Completeness is neutral. It is not possible to tell if the user's goals and expectations have been fully elicited, represented, and modeled from the solution space representations alone.

Table 2 gives some indication of the overall effect of each approach on project risk. Consider the possible consequences of developing a system based on a requirement definition that was either not verified or not validated. Both situations occur frequently. An un-validated, un-verified system is likely to be a complete failure. A verified, un-validated system will "work", but it will not meet the user's goals. This is termed a design failure because the user's requirements were incorrectly understood. An unverified, validated system may meet the user's goals, but is likely to be costly, unreliable, and defect prone. This is termed a technical failure, because the system was poorly built. A verified, validated system should meet the user's goals and be operationally effective.

Our assessment of formal and informal methods shows that with respect to validation, informal methods are neutral or enabling and formal methods are neutral. With respect to verification, informal methods are an obstacle or neutral and formal methods are enabling. Used in isolation, informal methods are likely to result in technical failures. Similarly, formal methods alone may lead to design failures. Using an appropriate mix of methods is more likely to result in a feasible, useful system.

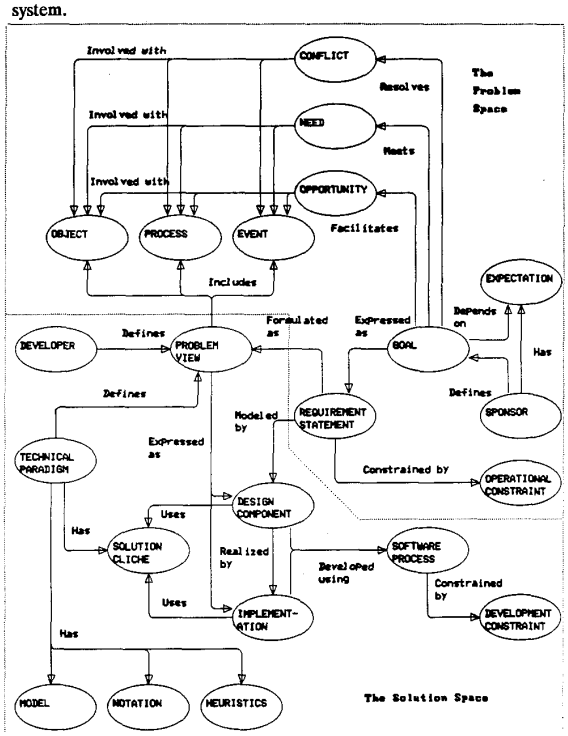

Figure 1 The Requirements Context Model

We do not mean to suggest that these outcomes are necessary results of either approach, or of specific verification and validation strategies. Neither do we argue that the approaches are sufficient conditions for the outcomes. We do mean to suggest that the approaches can strongly influence the outcomes, other things being equal.

No single conventional method for requirement definition provides complete coverage of the requirements context model, nor can it completely meet the goals of a good SRS. There are significant trade-offs in facilitating technical communication versus user communication. Project risk is minimized when an appropriate mix of methods is applied. However, even a well-crafted balance of formal and informal methods can be problematic, because only limited support for problem space representation can be provided. Given the complexity of present-day systems and the advisability of an eclectic approach, integrated support for methods and tools is a practical necessity.

TABLE 1 SRS GOALS AND REQUIREMENT DEFINITION APPROACHES

IEEE 830 SRS Characteristic	Effect of Informal Approaches	Effect of Formal Approaches
Unambiguous	Obstacle	Enables
Complete (Validation)		
No omissions	Neutral/Enables	Neutral
All user needs met	Neutral/Enables	Neutral
Minimal	Neutral/Enables	Neutral
Problem-oriented	Neutral/Enables	Neutral
Verifiable	Obstacle/Enables	Enables
Feasible	Obstacle/Enables	Neutral/Enables
Consistent		
No conflicts	Obstacle	Enables
No Structural errors	Obstacle	Enables
No Behavioral errors	Obstacle	Enables
No Protocol errors	Obstacle	Enables
Modifiable	Neutral/Obstacle	Enables/Obstacle
Traceable	Enables/Neutral	Neutral/Enables
Usable later	Enables	Neutral/Enables

TABLE 2
PROJECT RISK AND REQUIREMENT DEFINITION APPROACHES

	Informal Approach Used (Validate-able)	
	No	Yes
No Formal Approach Used (Verifiable)	Complete Failure (no method used)	Technical Failure (informal method only)
Yes	Design Failure (formal method only)	Successful System (eclectic approach)

IV. REQUIREMENT DEFINITION AUTOMATION

As part of KB/RMS development, automated support for requirement definition was reviewed. This survey is not exhaustive, but covered representative commercial and research systems. Table 3 provides an overview of the surveyed systems. Some are essentially data processing systems, in that they perform well-defined, albeit complex, processing on design representations. Some are knowledge-based, in that they rely on search and opportunistic processing. The only system that covers all aspects of the solution space and problem space is the KBSA proposal [Green 86]. Our analysis indicates research systems tend to focus on one aspect of representation, validation, or automated support. Commercial systems tend to focus on a specific need for a specific market segment. KB/RMS is intended to bridge some of these gaps.

4.1 Computer Aided Software Engineering. There are many commercially available Computer Aided Software Engineering (CASE) systems that provide some form of support for requirement definition. CASE tools typically support a generally accepted software design method, for example structured analysis and structured design. These methods may be used to model and represent software requirements within the solution space. Several CASE vendors use or are planning some form of expert-system technology in their products [Frenkel 85], [Desmond 87], [Bochenski 89]. This usage is sporadic, and does not reflect a serious attempt to apply knowledge-based approaches to CASE. A discussion of the state of the practice in automatic programming reaches a similar conclusion [Rich 88].

4.2 Knowledge-based Systems. Rich and Waters' characterization of the motivation for developer's assistants is instructive:

> "Most ... experimental programming tools ... seek to completely automate some aspect of the programming problem. However, because of the difficulties involved, most of these tools fail to reach this goal and require human guidance. ... There are basically three possible reactions to the need for human guidance: eliminate it by simplifying the tool, grudgingly tolerate it while waiting for advances in research, or intentionally design the tool to be an intelligent assistant." [Rich 86, p. 329]

The surveyed systems include both major types. We looked at automatic programming systems that accepted natural language input, and several developer's assistants. In particular, there are at least four knowledge-based requirement assistants: KBRA, KATE, the Analyst Assistant, and the Requirements Apprentice (see table 3 for references). A second group focused on expert-system approaches to specification production: ASDM, ASPIS, CASE/MVS, STES, and IDeA. The balance provide specialized support for a variety of software development tasks and approaches.

4.3 Coverage and Constraints. Table 3 lists some salient aspects of the surveyed systems. The Problem Space Coverage column indicates the extent to which the system represents the user's frame of reference. Solution Space Coverage indicates the extent of representation of reusable information to generate specifications or programs. Coverage of either aspect of the RCM was categorized as: 1) *Complete*, if the system represented most, if not all RCM objects; 2) *Partial*, if the system represented roughly half or more of the RCM objects; 3) *Minimal*, if the system represented three or fewer objects, or 4) *None*. A domain-specific system is explicitly constrained to a type of application or problem; a paradigm-specific system is explicitly constrained to a specification or code production method.

The SRS support entries indicate the extent to which the system support the SRS goals [IEEE 84]. The input and output columns describe what the user supplies and what the system produces in response. An asterisk in either the input or output column means the system accepts or produces system specific data, for example, input commands and query displays.

V. KB/RMS

5.1 Functional Description. The Knowledge-Based Requirements Management System (KB/RMS) has several major functions: *Input*, to enter requirement statements; *Augmentation*, to create new information by inferencing; *Inquiry*, for reporting and query on the database and knowledge representations; *Verification and Validation*, for automated consistency and completeness checking; *Generation*, for rule-based production of formal specifications from the database and knowledge-bases; and *Planning* for project sizing, versioning, and cost-benefit analysis. All of this must be preceded by knowledge acquisition.

Since many different functional sequences are typical in practice, no strict sequential input and output dependencies are present. Usage should be naturally regulating, since the utility of each function depends on the completeness of the requirements entered.

5.2 Knowledge Acquisition. Production and inference rules must be set-up for KB/RMS to operate. Representations of generic models, and a vocabulary for natural language recognition are needed as well. Ideally, these representations would be extensible without recourse to a programming language. There are several classes of rules that need to be represented: domain, paradigm, and RMS types. Knowledge about

these subject areas exists and can be defined to KB/RMS independent of any particular application project. Table 4 describes KB/RMS knowledge usage by subject area.

Domain knowledge covers specific subject areas like manufacturing, avionics, telecommunications, etc. With in each area, a generic system model would provide a list of commonly found functions. These lists could be built from several sources: published reports of actual systems, product descriptions of commercially available systems, and industry standards (IEEE, ANSI, NIST, ISO, etc.).

Paradigm knowledge concerns point of view, modeling semantics, and heuristics used in a particular technical paradigm. There are many published sources for this kind of information.

Ideally, both domain and paradigm knowledge would be made available in a software controlled library using a common representation. The library would be organized in a hierarchy of general and specific subjects. The library would be constructed by translating published sources into the common representation. Interviews and validation with experts would enhance and extend the library. Each component in the library would be associated with a node in the RMS type model. This would provide a mapping between domains and paradigms -- a sort of software Rosetta Stone.

The RMS type table is a representation of a third class of applicable knowledge: the generic structure of software systems [Binder 87]. This model provides a common point of reference for mapping generic domain and paradigm knowledge to a particular application project. It is essentially a taxonomy of generic components common to all software systems.

Table 3 Automated Support for Requirement Definition.

System/Source	Problem Space Coverage	Domain Specific ?	Solution Space Coverage	Paradigm Specific ?	SRS Support	Input	Output
Excelerator (Index 89)	None	No	Complete	Yes	Minimal	NL	NL *
REVS [ALford 85]	Minimal	Yes	Complete	Yes	Good	RSL *	R-Nets *
RMS [Binder 87]	Minimal	No	None	No	Partial	NL	NL *
NLP [Heidorn 86]	None	Yes	Partial	Yes	None	NL	GPSS
ISI [Heidorn 86]	Minimal	Yes	Partial	Yes	None	NL *	AP/1
MIT [Heidorn 86]	Minimal	No	Partial	Yes	None	NL	PL/1
IBM [Heidorn 86]	Complete	Yes	Minimal	No	None	NL	BDL
SAPE [Balzer 79]	Partial	Yes	Partial	Yes	None	NL *	AP2
CHI [Smith 85]	None	No	Complete	Yes	None	VHLL	object code
KBEmacs [Waters 85]	None	No	Partial	Yes	None	*	Ada, LISP
Designer Verifier [Moriconi 79]	None	No	Complete	Yes	Minimal	Gypsy	Gypsy, *
IDeA [Lubars 89]	None	No	Complete	Yes	Minimal	SA/SD	SA/SD, *
Inscape [Perry 89]	None	No	Complete	Yes	None	*	*
KBRA [Czuchry 88]	Partial	Yes	Minimal	No	Minimal	NL	*
KATE [Ficas 88]	Complete	No	None	No	Partial	Petri-net	*
Analyst Assist [Loucopoulos 89]	Complete	No	Partial	Yes	Minimal	NL	JSD
Rqt Apprentice [Rubenstein 89]	Complete	No	Minimal	No	None	NL	*
Impulse 86 [Schoen 88]	Partial	Yes	Partial	Yes	None	*	Impulse 86
ASDM [Blackburn 89]	None	No	Complete	Yes	Minimal	*	Refine
ASPIS [Puncello 88]	Minimal	No	Partial	Yes	Minimal	*, SADT	*
CASE/MVS [Symonds 88]	None	No	Partial	Yes	Minimal	*	P-code
STES [Tsai 88]	None	No	Partial	Yes	None	SA, *	SD, *
SMILE/MARVEL [Kaiser 88]	None	No	Partial	No	None	*	*
KBSA [Green 86]	Complete	No	Complete	Yes	Yes	NL *	*

Note: SA = Structured Analysis, SD = Structured Design, JSD = Jackson Structured Design, SADT = Structured Analysis and Design Technique; NL = Accepts/Produces natural language, * = system specific input or output.

5.3 Input. Knowledge acquisition for a specific project is necessary. The analyst must be able to enter requirement statements. At a minimum, conventional keyboard text entry with full-screen text edit features is needed. This function checks spelling and basic syntax, stores the requirement text, associates the text with static entities (user, date supplied, cost/benefit estimate, etc.), facilitates creation of hypertext nodes and links, and extracts semantic units to be stored in the RCM.

Upon input, each requirement statement is parsed to extract semantic units. These units are placed in a conceptual graph model. A semantic unit consists of a qualitative relationship among several objects. A semantic unit is inferred from the syntax of the requirement statement and clues provided by keywords. The objects and relationships in a semantic unit are added to the conceptual graph, if not already present in it. Only the new parts of the semantic unit are added. The conceptual graph is initially empty, and is populated by extracting semantic units as each requirement statement is input. Upon each input or change to a requirement statement, the conceptual graph is retrieved and updated.

TABLE 4 KB/RMS KNOWLEDGE REPRESENTATION AND USAGE

Function Using	Rule Subject Area		
	Domain knowledge	Paradigm knowledge	RMS Type model
Augment	Reuse generic and specific requirements		Interpret requirement statements
Verify and Validate	Check coverage of generic domain model against user requirements	Check sufficiency of requirements for specification generation	Check coverage of RMS type model by user requirements
Generation	Support translation	Translate specific objects into paradigm specific representations	Support translation

5.4 Augmentation. The input function creates the RCM by parsing and semantic interpretation. Pragmatic interpretation is used to map objects in the RCM onto predetermined paradigm and domain models. The three phases of natural language processing create two types of new information about the requirements. The first type is *intrinsic*, because new information is obtained from a systematic representation of the input without recourse to other semantic sources. The second type is called *augmented*, because it is synthesized from intrinsic information and generic knowledge.

The transformation of natural language to a conceptual graph model creates information not present in the original requirement statements. The new information is about connections between objects referenced in separate statements. For example, the model would represent all the linkages between "book" and "author". With a large number of requirement statements (say, more than 100) knowledge of all contexts in which an object is referenced can only be obtained with considerable human effort. An analyst can become aware of the connections by interviews, direct participation, or careful study of requirement statements. It is difficult to assess, either subjectively or objectively, when a sufficient level of understanding has been reached. When more than one analyst becomes involved, each person's understanding is necessarily different. With large projects, division of labor among analysts means that it is unlikely that any one person will have a complete grasp of all requirements. KB/RMS can readily extract, represent, and maintain the context of all requirements and semantic units.

Augmentation is the process of linking intrinsic information with generic models and expert knowledge about paradigms and domains. The assignment of types to requirement statements is a clear example. Several modes of augmentation are envisioned.

Typing will be initiated by the analyst. The analyst will assign types from the RMS type table to each requirement. This will allow KB/RMS to map between the domain, paradigm, and application semantic models.

Rule driven augmentation will be requested by the analyst. It will evaluate each requirement statement, and prompt the analyst to supply missing information or clarify inconsistencies. The rules will be based on expert knowledge about requirement definition, domains, and paradigms. An "explaining" facility that gives the inference trace for each prompt will

assist the analyst in making good responses. For example, KB/RMS would search for a semantic unit typed as a function, and which described the highest level of aggregation. The analyst would be prompted to designate sub-process. Suggestions would be made from clichés and templates. Each RMS type would have a standard prompt list. The list would be composed from material the analyst entered, clichés, inferences, and from templates taken from the KB/RMS model.

Rule driven augmentation could also be non-interactive. For some kinds of analysis, deterministic processing could be used. For example, an incidence matrix of all objects could be checked for missing data threads, and new objects and associations could be generated. The system would also create a corresponding synthetic text requirement. The synthetic requirement would have a trace of the rules that lead to its creation.

5.5 Inquiry. KB/RMS will use a relational DBMS to support ad hoc inquiry. An "open architecture" concept will allow access to data in machine readable form. Special facilities will need to be developed for inquiries on the knowledge base. This would include text and graphic output of the RCM, an "explainer" for the augmentation and generation facilities, and a means to link the requirement definition DBMS and the knowledge base. Additional pro-forma output capabilities should also be provided.

5.6 Validation and Verification. The size of natural language descriptions for present-day systems makes automated support for requirements validation and verification necessary. At a minimum, the capability to check for each of the desirable characteristics of an SRS (see Table 1) is needed.

Minimal Ambiguity. By creating an explicit model of the problem and solution spaces, ambiguity will be reduced. The reduction of text to a conceptual graph will provide the analyst with another point of view on the requirements. Some simple editing (as provided by commercially available grammar checking programs), and prompting for replacement of vague terms would also reduce ambiguity.

Necessary and Sufficient Solution. This is validation, or determining whether the system is an adequate solution to the user's problem. There are four major categories of validation: 1) absence of omissions, 2) coverage of all user needs, 3) minimal (no "gold-plating"), 4) problem-oriented (no unnecessary technical experiments).

By representing the problem space as well as the solution space, the data exists to check coverage of all relevant problem space objects. The inquiry capability will provide many views of the requirements and the RCM. This should aid the process of review by the analysts and users. It cannot eliminate the need for human review, but it can aid the process.

By representing domain information and common solutions within a domain, KB/RMS could prompt for specific requirements, or automatically supply them. Reuse of existing validated requirements, if stored in a suitable library, could also improve consistency and coverage.

Amenable to Verification. By using the generic system model template, KB/RMS can check for the presence and sufficiency of requirements from a structural point of view. For example, does the model include input and output descriptions? A data thread analysis can identify requirements that call for output not described as an input or generated as the result of a process.

Technical and Economic Feasibility. The planning function of KB/RMS will include cost analysis and version identification functions. This will assist in creating better plans and managing project risk.

Internal Consistency. A consistent requirement definition has several characteristics. The meaning and usage of terms is consistent. There are no incorrect or missing linkages between inputs and outputs. Descriptions of the system's behavior should not be redundant, ambiguous, or conflicting. This goal is supported by checking each statement as it is entered, and providing the analyst with feedback about explicit and implicit relationships in the model.

Simulation of the solution space model is not considered a part of KB/RMS. Since there are existing systems that provide this kind of analysis, KB/RMS would be used to create a formal specification suitable for simulation. The translation process would be supported by a selection from the paradigm library.

Ease of Modification. For a large collection of narrative requirement statements without any sort of indexing, making a change presents a problem similar to adding a new character to a novel or movie. One needs to grasp the entire "plot" and "cast of characters" to make a revision that maintains the story and style. KB/RMS will provide indexing, keyword search, and concordances on the text. The context and source of each semantic unit is preserved. With this information and related automated support, the mechanical aspects of modifying a requirement definition become simple.

Design and Implementation Traceability. Because each requirement is tagged and production of design representations is automated, KB/RMS can readily support traceability. KB/RMS supports traceability for the same reasons it supports modification: each requirement is tagged to provide indexing, keyword search, and concordances.

Suitability for Software Testing and Maintenance. A testable requirement has a corresponding implementation that either meets the requirement or does not. This means the requirement must describe an objective result, feature, condition, or behavior. This result must be identifiable and repeatable for the requirement to be testable. KB/RMS encourages concise, unambiguous requirements and provides many ways to tag a requirement and its context. This makes the identification of acceptance criteria straight-forward. The representation of related problem space information (goals and constraints) also provides information for test case development.

5.7 Generation. The concept of generating formal specifications from the RCM is based on an observation of how system developers produce specifications. The problem space model is mentally built by the analyst. It is not explicit, even in most formal methods. The analyst performs cognitive processing to place new facts into this model as the users are interviewed and the operational environment is researched. The analyst also must know, develop, or intuit transformation rules from his problem space model to the solution model.

KB/RMS Generation is an enhancement of RMS modeling support [Binder 87]. As a first step in developing a data flow model and an entity-relationship model of a system with a large set of requirements, all requirements assigned a type of "process" or "entity" were listed. The analyst scanned the list. Words that seemed to be good candidates for processes, data flows, data stores, entities, or relationships were marked with a colored highlighting pen. A quick sketch of the solution model was developed from these words, and then entered into a CASE system.

KB/RMS will automate much of this process. The RCM will be instantiated by parsing, semantic interpretation, and augmentation. This automates the first step: abstracting the relevant problem space objects. By representing expert knowledge in rules that can work on the RCM, we can automate the second part of the process: sketching the formal representation and entering it into a CASE system.

The input, augmentation, and validation and verification processes will each incrementally build information about the requirements. For example, a requirement states:

Provide a function to add a book to the database.

The RMS system allows the developer to assign types (components of a software system meta-model) to each requirement. This mapping between the problem and solution spaces is also used in KB/RMS. Suppose the following types had been assigned to the above requirement:

Process, transformation, maintain
Data, entity

This identifies the requirement as describing a process that transforms input to maintain a database. We can represent the information provided by the types with a conceptual graph [Sowa 84]. The concept name is given inside square brackets: [concept]. Conceptual relations are enclosed in parenthesis: (relation). The arrows indicate the direction, or sense, of the relation.

[system]→(has)→[function]→(isa)→[add]→(object_isa)→[book]
[add]→(has_type)→[maintain]
[book]→(has_type)→[entity]

The type table inheritance would be represented as:

[Process]→(has_subtype)→[transformation]→(has_subtype)→[maintain]
[Data]→(has_subtype)→[entity]

We can use this representation and some production rules to create outputs that can be directly entered into a CASE system. The following rules are given a pseudo-code for an expert-system shell:

For each [<node_name>] that (has_type)→[entity]
Create a CASE entity record for [<node_name>]
Create a CASE data store record [<node_name>];

For each [<node_name>] that (has_type)→[process]
Create a CASE process record [<node_name>];

For each [<node_name>] that (has_type)→[process] and
(has_type)→[entity]
Create a CASE data flow record for [<node_name>]
with endpoint_1 = [<node_name>]
with endpoint_2 = [<entity_name>];

VI. Knowledge Representation

6.1 Natural Language and Semantic Modeling. Representation for a grammar and vocabulary is straightforward in PROLOG. Semantic interpretation is significantly more difficult. [Balzer 79] identifies a number of specific problems in machine interpretation of natural language specifications.

We address the ambiguity problem in two ways. First, we assume requirement statements describe the problem space. Second, we look only for subject-action-object units. Third, given a large enough problem, we hope that correct interpretations outweigh incorrect ones, so we have an analog to a central tendency.

The analyst is responsible for stating things clearly to begin with. Remaining ambiguity can be revealed by analysis of the RCM. At each successive refinement, the system model can be cooperatively improved by the analyst and KB/RMS. The intent is not to produce a perfect representation, but a usable one.

There is no direct limitation on domain. The system is vocabulary and grammar constrained, in that it requires rules to specify vocabulary, parts of speech, and acceptable syntax. Recognizing a new domain means adding its vocabulary, heuristics, and clichés.

6.2 Representing Expert Knowledge. KB/RMS incorporates subjective heuristics for augmentation, validation, verification, and generation. There are at least three general classes of relevant expert knowledge: process, domain, and paradigm. Process knowledge has to do with how to go about requirement definition. Many of our heuristics for validation are process heuristics. Domain knowledge has to do with experience about solutions for similar problems. For example, if we are building an avionics system, it would be useful to have a checklist of functions from other existing avionics systems. This is a domain cliché. Paradigm knowledge has to do with how to construct the solution model from the problem space information. The cliché and plan concept is a powerful example of paradigm knowledge. There are many possible sources for these heuristics.

Process Heuristics. [Gause 89], for example, suggests using "context-free" questions to explore the problem space. These could readily be incorporated into a prompting dialogue with the analyst, and the answers stored in conceptual graph representation.

Domain Heuristics and Clichés. A useful approximation of domain knowledge would be to develop functional check-lists from existing systems. These check-lists would consist of objects and processes in these systems, mapped onto the RMS type model. As the requirements for a new system were entered, the analyst could "borrow" the generic domain model, or compare the evolving model to the domain model. It would be useful to have a generic domain model, with each example system as a sub-type with inheritance. This could be used to support validation and augmentation. Borrowing from the domain model could generate synthetic requirements.

Paradigm Heuristics. There are many heuristics commonly used to create system models from requirements. As with the process and domain rules, these kind of heuristics can be used for prompting and augmentation. Paradigm heuristics can also be used in generation. For example, if we had several semantic units that were typed as data attributes, and were associated with several objects typed as data entities, we could infer an is-a hierarchy.

[Object_1]→(is_type)→[entity]
[Object_2]→(is_type)→[entity]

[Object_3]→(is_type)→[attribute]
[Object_4]→(is_type)→[attribute]

[Object_3]→(part_of)→[Object_1]
[Object_6]→(part_of)→[Object_1]

Given this representation and a super-type discovery heuristic, we can formulate rules to create outputs that can be directly entered into a CASE system. The following rules are given a expert-system shell pseudo-code:

For each [<node_name>] that
(has_type)→[entity] and
(has_type)→[attribute] and
[attribute]→(part_of)→[<different_node_name>]

Create a CASE entity record [<node_name>]
Create a CASE entity record [<different_node_name>]
Create a CASE relationship record [<different_node_name>],
[<node_name>], <rel=sub-type>;

VII. Conclusions and Future Research

Due to the complex and ambiguous nature of the requirements definition process, it is necessary to have a common model which can be used to represent and process various entities, concepts, conceptual relations, and objects in an integrated manner. We have presented a framework and system model for an intelligent assistant for requirement definition in this paper. A prototype system has been implemented that uses the vocabulary from the library problem [Wing 88]. Since KB/RMS is independent of any particular application domain and development method, we plan to develop a complete system in the future.

References

[Alford 85] Alford, Mack. "SREM at the Age of Eight: The Distributed Computing Design System." *IEEE Computer*, April 1985, pp. 36-46.

[Balzer 79] Balzer, R., Goldman, N., and Wille, D. "Informality in Program Specifications." *Artificial Intelligence and Software Engineering.*, edited by Charles Rich and Richard C. Waters: Los Altos, Ca.: Morgan Kaufmann Publishers, Inc., 1986.

[Binder 87] Binder, Robert, *Requirements Management System: User Guide.* Chicago: Robert Binder Systems Consulting, Inc., 1987.

[Bochenski 89] Bochenski, Barbara. "Declaring the Facts of the Inference Difference", *Software Magazine*, May 1989.

[Czuchry 88] Czuchry, Andrew J. and Harris, David R. "KBRA: A New Paradigm for Requirements Engineering," *IEEE Expert*, Winter 1988, v3:21-35.

[Davis 88] Davis, Alan M.,"A Comparison of Techniques for the Specification of External System Behavior," *Communications of the ACM*, September 1988, v31: 1098-1115.

[Desmond 87] Desmond, John, "Expert Systems in Software Development," *Software Magazine*, July 1987.

[Fickas 88] Fickas, Stephen F., and Nagarajan, P. "Critiquing Software Specifications,", *IEEE Software*, November 1988, 37-47.

[Frenkel 85] Frenkel, Karen A.,"Towards Automating the Software-Development Cycle,", *Communications of the ACM*, June 1985, v.28, n. 6, pp. 578-589.

[Gause 89] Gause, Donald C., and Weinburg, Gerald M. *Exploring Requirements: Quality Before Design,* New York: Dorset House Publishing Co., 1989.

[Green 86] Green, C., Luckham, D., Balzer, R., Cheatham, T., and Rich, C. "Report on a Knowledge-Based Software Assistant." *Artificial Intelligence and Software Engineering.*, edited by Charles Rich and Richard C. Waters: Los Altos, Ca.: Morgan Kaufmann Publishers, Inc., 1986.

[Heidorn 86] Heidorn, G.,"Automatic Programming Through Natural Language: A Survey," in *Artificial Intelligence and Software Engineering.*, edited by Charles Rich and Richard C. Waters: Los Altos, Ca.: Morgan Kaufmann Publishers, Inc., 1986.

[IEEE 84] *ANSI/IEEE Standard 803-1984: Standard for Software Requirements Specifications,* New York: The Institute of Electrical and Electronic Engineers, 1984.

[Index 89] Index Technology Corporation, *Excelerator/IS: Data & Reports Reference Guide, Release 1.9.* Cambridge, Ma: Index Technology Corporation, 1989.

[Kaiser 88] Kaiser, Gail E., Feiler, Peter H., and Popovich, Steven S., "Intelligent Assistance for Software Development and Maintenance," *IEEE Software*, May 1988.

[Loucopoulos 89] Loucopoulos, P., Champion, R. E. M., and Harthoorn, C. "Knowledge-Based Support for Requirements Acquisition." *Proceedings: STA 5, Fifth Conference of the Structured Techniques Association*, May 1989, 295-311.

[Lubars 89] Lubars, Mitchell D. "The IDeA Design Environment." *Proceedings: 11th International Conference on Software Engineering.* Washington, D.C.: IEEE Computer Society Press, 1989.

[Moriconi 79] Moriconi, Mark S. "A Designer/Verifier's Assistant." *IEEE Transactions on Software Engineering*, July 1979, v.5:pp. 387-401.

[Perry 89] Perry, Dewayne E. "The Inscape Environment." *Proceedings: 11th International Conference on Software Engineering*, Washington, D.C.: IEEE Computer Society Press, 1989.

[Puncello 88] Puncello, P.P., Torrigiani, P., Pietri, F., Burlon, R., Cardile, B, and Conti, M., "ASPIS: A Knowledge-Based CASE Environment," *IEEE Software*, March 1988, 58-65.

[Reubenstein 89] Reubenstein, H.B. and Waters, R.C. "The Requirements Apprentice: An Initial Scenario," *Proceedings: Fifth International Workshop on Software Specification and Design*, [special issue of ACM SIGSOFT Software Engineering Notes], May 1989, v14:211-218.

[Rich 86] Rich, Charles. "A Formal Representation for Plans in the Programmer's Apprentice," *Artificial Intelligence and Software Engineering.*, edited by Charles Rich and Richard C. Waters: Los Altos, Ca.: Morgan Kaufmann Publishers, Inc., 1986.

[Rich 88] Rich, Charles, and Waters, Richard C. "Automatic Programming: Myths and Prospects," *IEEE Computer*, August 1988, pp. 40-52.

[Schoen 88] Schoen, E., Smith, R.G., and Buchanan, B.G. "Design of Knowledge-Based Systems with a Knowledge-Based Assistant," *IEEE Transactions on Software Engineering*, December 1988, 14:1771-1791.

[Smith 85] Smith, D. R., Kotik, G. B., and Westfold, S. J. "Research on Knowledge-Based Software Environments at Kestrel Institute," *IEEE Transactions on Software Engineering.* November 1985, 11:1278-1295.

[Symonds 88] Symonds, Andrew J. "Creating a Software Engineering Knowledge Base," *IEEE Software*, March 1988, 50-57.

[Tsai 88] Tsai, Jeffery J-P. and Ridge, Joel C. "Intelligent Support for Specifications Transformation," *IEEE Software*, November 1988, 28-35.

[Waters 85] Waters, Richard C. "The Programmer's Apprentice: A Session with KBEmacs," *IEEE Transactions on Software Engineering.* November 1985, 11:1296-1320.

[Wood 89] Wood, David P., and Wood, William G. *Comparative Evaluations of Four Specification Methods for Real-Time* Technical Report CMU/SEI-89-TR-36, 1989, Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, December 1989.

[Wing 88] Wing, Jeannette M. " A Study of 12 Specifications of the Library Problem," *IEEE Software*, July 1988, 66-76.

[Yaverbaum 89] Yaverbaum, Gayle. "Specifying System Requirements: A Framework of Current Techniques," *Journal of Information Systems Management*, Winter 1989, v. 6, n. 1, pp. 17-21.

[Yeh 84] Yeh, R.T, Zave, P., Conn, A.P., and Cole, G.E. "Software Requirements: New Directions and Perspectives," *Handbook of Software Engineering.*, edited by Charles R. Vick and C. V. Ramamoorthy: New York: Van Nostrand Reinhold Company, Inc., 1984.

[Zave 82] Zave, Pamela. "An Operational Approach to Requirements Specification for Embedded Systems," *IEEE Transactions on Software Engineering*, May 1982, 8:250-269.

[Zultner 89] Zultner, Richard E. "Objectives: The Missing Piece of the Requirements Puzzle." *Proceedings: STA 5, Fifth Conference of the Structured Techniques Association.* May 1989, 283-294.